CS 453X: Class 5

Jacob Whitehill
Example 2: weather data

- Data from https://www.kaggle.com/selfishgene/historical-hourly-weather-data/data

- Hourly measurements of pressure, humidity, and temperature, and wind speed for different cities in the USA and Israel.

- At each time $t$, how accurately can we predict temperature at time $t+1$ hour in Boston?
Linear regression

- Linear regression is one of the few ML algorithms that has an analytical solution:

\[ w = (XX^\top)^{-1} Xy \]

- **Analytical solution**: there is a closed formula for the answer.
Linear regression

- Alternatively, linear regression can be solved numerically using gradient descent.

- **Numerical solution**: need to iterate (according to some algorithm) many times to *approximate* the optimal value.

- Gradient descent is more laborious to code than the one-shot solution, but it generalizes to a wide variety of ML models.
Linear regression in 1-d

- Let’s look at a simple 1-d example of linear regression again...

- Here are some different weights $w$ and associated costs ($f_{\text{MSE}}$):
Linear regression in 1-d

• Let’s look at a simple 1-d example of linear regression again...

• ...and here is the graph of the function $f_{\text{MSE}}(w)$:
Finding the best $w$

- Why not just “jump” to the optimal $w$ that minimizes $f_{MSE}$?
Finding the best $w$

• Why not just “jump” to the optimal $w$ that minimizes $f_{MSE}$?

• In 1-d, we actually could:

  • Just sample many $w$ values:

    $w$  
    
    -3  -2.99  -2.98  ...  5.99  6

  • Compute $f_{MSE}$ for each possible $w$.

    $f_{MSE}(w)$  
    
    ...  ...  ...  ...  ...  ...

• Pick the best one.
Finding the best $w$

- But what about in higher dimensions (e.g., 2-d).

- We could search through all possible combinations of $(w_1, w_2)$:

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3, -3</td>
<td>-3, -3</td>
</tr>
<tr>
<td>-2.99, -3</td>
<td>-2.99, -2.99</td>
</tr>
<tr>
<td>5.99, -3</td>
<td>5.99, -2.98</td>
</tr>
<tr>
<td>6, -3</td>
<td>6, -2.99</td>
</tr>
</tbody>
</table>

![Graph showing a 3D surface representing $f_{true}(w_1, w_2)$]
Curse of dimensionality

- As the number of dimensions $m$ increases, so does the number of possible values for $w$ that we have to probe.

- If we want to sample 100 values per dimension, then we have $100^m$ values for $m$ dimensions.

- For a 24x24 image, we have $m=576$ dimensions $\implies 100^{576}$

- Completely infeasible.
Gradient descent

- Gradient descent is a **hill climbing algorithm** that uses the gradient (aka slope) to decide which way to “move” $w$ to reduce the objective function (e.g., $f_{\text{MSE}}$).
Gradient descent

• Suppose we just guess an initial value for $w$ (e.g., -2.1).

• How can we make it better — increase it or decrease it?
Gradient descent

- Suppose we just guess an initial value for \( w \) (e.g., -2.1).

- How can we make it better — increase it or decrease it?

- What does the slope of \( f_{\text{MSE}} \) tell us to do?

The slope at \( f_{\text{MSE}}(-2.1) \) is negative, i.e., we can decrease our cost by increasing \( w \).
Gradient descent

• Or maybe our initial guess for $w$ was 3.9.

• How can we make it better — increase it or decrease it?

• What does the slope of $f_{\text{MSE}}$ tell us to do?

The slope at $f_{\text{MSE}}(3.9)$ is positive, i.e., we can decrease our cost by decreasing $w$. 
Gradient descent

• How do we know the slope? Compute the gradient of \( f_{\text{MSE}} \) w.r.t. \( \mathbf{w} \):

\[
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \nabla_{\mathbf{w}} \left[ \frac{1}{2n} \sum_{i=1}^{n} (\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)})^2 \right] \\
= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[ (\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)})^2 \right] \\
= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)}) \\
= \frac{1}{n} \mathbf{X} (\mathbf{X}^\top \mathbf{w} - \mathbf{y})
\]
Gradient descent

• How do we know the slope? Compute the gradient of $f_{\text{MSE}}$ w.r.t. $w$:

$$
\nabla_w f_{\text{MSE}}(y, \hat{y}; w) = \nabla_w \left[ \frac{1}{2n} \sum_{i=1}^{n} (x^{(i)\top} w - y^{(i)})^2 \right]
$$

$$
= \frac{1}{2n} \sum_{i=1}^{n} \nabla_w \left[ (x^{(i)\top} w - y^{(i)})^2 \right]
$$

$$
= \frac{1}{n} \sum_{i=1}^{n} x^{(i)} (x^{(i)\top} w - y^{(i)})
$$

$$
= \frac{1}{n} X (X^\top w - y)
$$

• Then plug in the current value of $w$. 
  (Note that $X$ and $y$ are computed from the data and are constant.)
Gradient descent

- How far do we “move” left or right?

- Notice that, in the graph below, the magnitude of the slope (aka gradient) gives an indication of how far we need to go to reach the optimal $w$. 

\[
\nabla_w f_{\text{MSE}}(-3.0) = -4.53 \\
\n\nabla_w f_{\text{MSE}}(0.7) = -0.65
\]
Gradient descent algorithm

• Set $\mathbf{w}$ to random values; call this initial choice $\mathbf{w}^{(0)}$. 
Gradient descent algorithm

- Set $\mathbf{w}$ to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
Gradient descent algorithm

- Set \( \mathbf{w} \) to random values; call this initial choice \( \mathbf{w}^{(0)} \).
- Compute the gradient: \( \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)}) \)
- Update \( \mathbf{w} \) by moving opposite the gradient, multiplied by a step size \( \varepsilon \).

\[ \mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)}) \]
Gradient descent algorithm

• Set $w$ to random values; call this initial choice $w^{(0)}$.
• Compute the gradient: $\nabla_w f(w^{(0)})$
• Update $w$ by moving opposite the gradient, multiplied by a step size $\varepsilon$.  
  \[ w^{(1)} \leftarrow w^{(0)} - \varepsilon \nabla_w f(w^{(0)}) \]
• Repeat…  
  \[ w^{(2)} \leftarrow w^{(1)} - \varepsilon \nabla_w f(w^{(1)}) \]
  \[ w^{(3)} \leftarrow w^{(2)} - \varepsilon \nabla_w f(w^{(2)}) \]
  
  \[ \ldots \]
  \[ w^{(t)} \leftarrow w^{(t-1)} - \varepsilon \nabla_w f(w^{(t-1)}) \]
Gradient descent algorithm

- Set $\mathbf{w}$ to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update $\mathbf{w}$ by moving opposite the gradient, multiplied by a step size $\varepsilon$. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Repeat…
  $\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$
  $\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$
  ...
  $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \varepsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$
- …until convergence:
  $| f(\mathbf{w}^{(t-1)}) - f(\mathbf{w}^{(t)}) | < \delta$

$\delta$ is a chosen convergence tolerance.
Gradient descent demos

- 1-d
- 2-d